QADA®

Quality Evaluation by QADA

Eila Niemelä & Mari Matinlassi
VTT Technical Research Centre of Finland

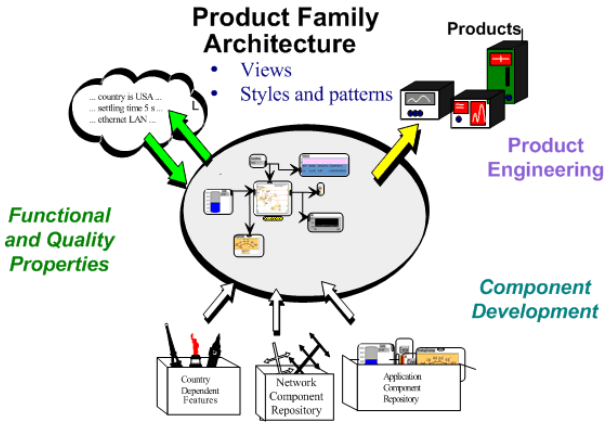
E-mail: {Eila.Niemela, Mari.Matinlassi}@vtt.fi

QADA®: http://www.vtt.fi/qada/

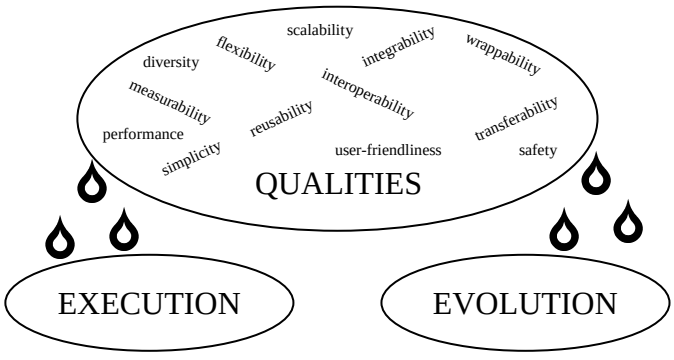
TUTORIAL OUTLINE

- INTRODUCTION
  - Main concepts of QADA®
- CAPTURING AND MAPPING QUALITY REQUIREMENTS TO ARCHITECTURE
- REPRESENTING QUALITIES IN ARCHITECTURE DESIGN
- EVALUATING EXECUTION QUALITIES
  - RAP method
- EVALUATING EVOLUTION QUALITIES
  - IEE method
- (RE)USING EXISTING DESIGN KNOWLEDGE
- CONCLUSIONS

PRODUCT FAMILY ENGINEERING



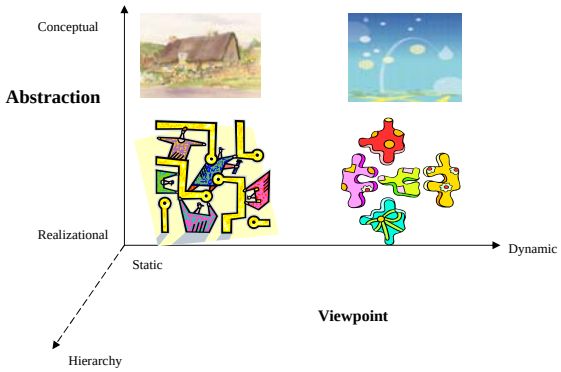
QUALITY ATTRIBUTES



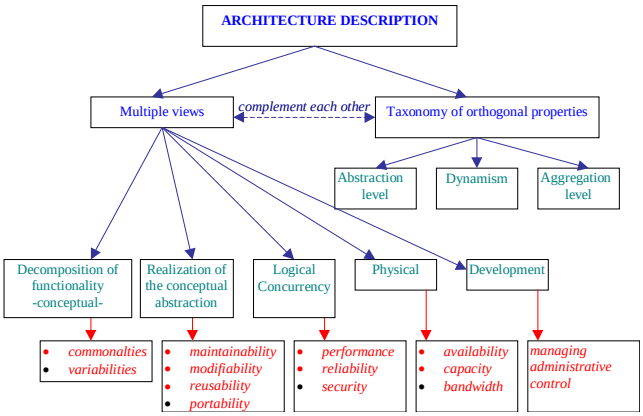
- performance
- security
- availability
- usability
- scalability
- reliability
- interoperability
- adaptability

- maintainability
- flexibility
- modifiability
- extensibility
- portability
- reusability
- integrability
- testability

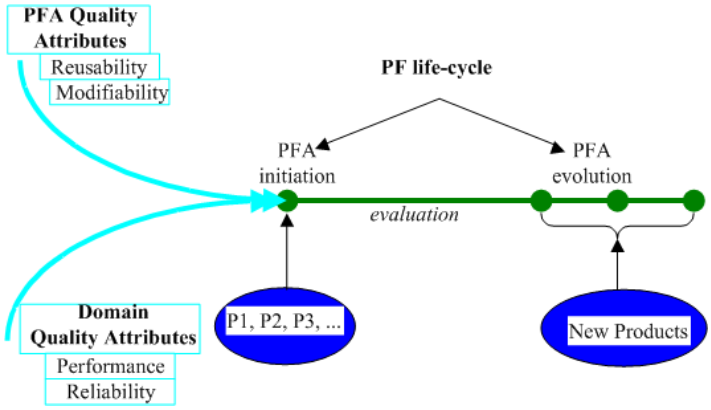
ORTHOGONAL PROPERTIES OF SOFTWARE ARCHITECTURE



SOFTWARE ARCHITECTURE & QUALITY

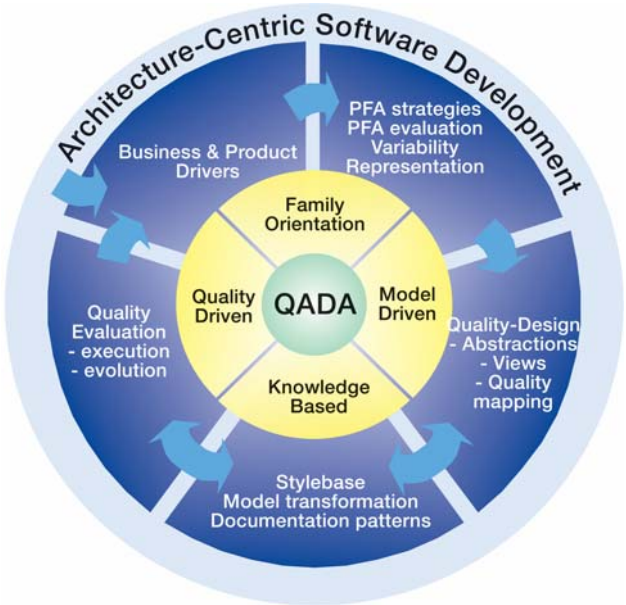


QUALITY OF PF LIFE CYCLE



QADA®

- Development started in 2000
- Industrial applications:
- Middleware services
- Distribution platforms
- Wireless services
- Wireless terminals
- Control systems
- Measurement systems
- Product families of embedded systems and software systems



TUTORIAL OUTLINE

- INTRODUCTION
  - Main concepts of QADA®
- CAPTURING AND MAPPING QUALITY REQUIREMENTS TO ARCHITECTURE
- REPRESENTING QUALITIES IN ARCHITECTURE DESIGN
- EVALUATING EXECUTION QUALITIES
  - RAP method
- EVQLUATING EVOLUTION QUALITIES
  - IEE method
- (RE)USING EXISTING DESIGN KNOWLEDGE
- CONCLUSIONS

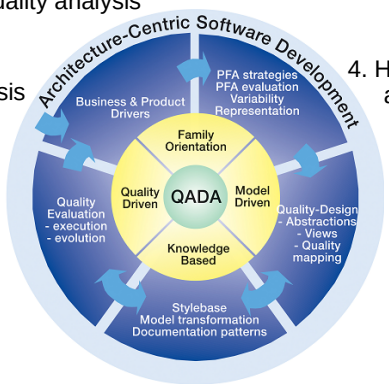
CAPTURING AND MAPPING QRs TO ARCHITECTURE

2. Quality analysis

3. Variability analysis

1. Impact analysis

4. Hierarchical domain analysis



1. IMPACT ANALYSIS

Goal:

To define the interested stakeholders and their targets concerning the product family

- **The concerns of different stakeholders are negotiated to achieve all relevant functional and quality requirements of a product family**
- **I*framework is used for requirements definition and negotiation**
  - **I* framework enables to describe dependencies and conflicts between stakeholders' concerns**
  - **Example**

2. QUALITY ANALYSIS

Goal:

To express quality requirements (QR) in a way that they can later be traced and measured

- **QAs must be prioritized**
  - **Requirements on the highest priority level have always to be met in architecture (to be considered in trade-off analysis)**
- **Evaluation criteria are derived from the QRs and classified to evaluation levels, e.g.:**
  - **Family specific QRs of**
    - **high priority**
    - **medium priority**
    - **low priority**
  - **System/domain specific QRs of**
    - **high priority**
    - **medium priority**
    - **low priority**

3. VARIABILITY ANALYSIS

Goal:

To define the QRs that vary on the business domain or stakeholders

Types of variability:

- **Variability among quality attributes.**
  - **For example, for one family member the reliability is important, but for other family members there are no reliability requirements.**
- **Different priority levels in quality attributes.**
  - **For example, for one family member the extensibility requirements are extremely high, whereas for others those requirements are at the lower level.**
- **Indirect variation**
  - **Functional variability can indirectly cause variation in the quality requirements or vice versa**

4. HIERARCHICAL DOMAIN ANALYSIS

Goal:

To map common and variable QAs to hierarchical service categories

- **The QRs common to all family members must be mapped to the common functionality of the family**
- **The architect has to decide which services are responsible for each quality requirement (scoping)**
- **One requirement may be mapped to several functional services (dependency mgmt)**
- **The quality requirements themselves may result to certain functionality (i.e. execution QAs)**
- **The requirements mapping is a specific work of the software architects and requires an extensive knowledge of the product family and its members**
- **Example**

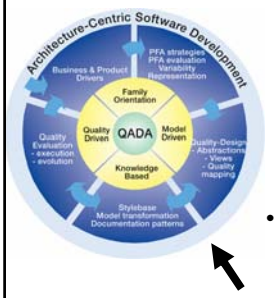
TUTORIAL OUTLINE

- INTRODUCTION
  - Main concepts of QADA®
- CAPTURING AND MAPPING QUALITY REQUIREMENTS TO ARCHITECTURE
- REPRESENTING QUALITIES IN ARCHITECTURE DESIGN
- EVALUATING EXECUTION QUALITIES
  - RAP method
- EVQLUATING EVOLUTION QUALITIES
  - IEE method
- (RE)USING EXISTING DESIGN KNOWLEDGE
- CONCLUSIONS

QUALITIES IN ARCHITECTURE

There are two main means to represent quality requirements in architecture:

1. The use of architectural styles and patterns
   - Styles and patterns employ qualitative reasoning to motivate when and under what conditions they should be used
2. The use of qualitative constrains, e.g. specific quality profiles
   - Profiles can be defined to extend the architectural models to support certain quality aspects



QUALITY REPRESENTATIONS: styles and patterns

- **Candidate architectural styles must be identified. For each style, it is examined how it meets the quality requirements**
- **Possible conflicts between QRs are identified and the trade-off analysis is carried out**
  - **NFR (Non-Functional Requirements) framework or Stylebase can be utilized in style selection and in conflicts solving**
- **The architectural style that meets the QRs best is then selected**

QUALITY REPRESENTATIONS: quality profiles

- UML 2.0 enables the description of all the viewpoints of QADA
- UML notation can be extended to support certain quality attributes using UML's own extension mechanism; profiles
  - A profile consists of stereotypes, tagged definitions and constraints
  - By creating a new stereotype, defining tags for it, and denoting the stereotype to extend the desired meta-class, the certain elements in architecture can be extended with a new profile
  - Profiles enable the attachment of quality properties to the architectural models

TUTORIAL OUTLINE

- INTRODUCTION
  - Main concepts of QADA®
- CAPTURING AND MAPPING QUALITY REQUIREMENTS TO ARCHITECTURE
- REPRESENTING QUALITIES IN ARCHITECTURE DESIGN
- EVALUATING EXECUTION QUALITIES
  - RAP method
- EVQLUATING EVOLUTION QUALITIES
  - IEE method
- (RE)USING EXISTING DESIGN KNOWLEDGE
- CONCLUSIONS

EVALUATING EXECUTION QUALITIES – RAP (Reliability & Availability Prediction) method

- **Introduction**
- Method overview
- Phases
  1. Defining reliability and availability goals
  2. Representing reliability and availability in architectural models
  3. Evaluating reliability and availability
- Case example

RAP INTRODUCTION

- An integrated part of QADA, extending it with reliability and availability (R&A) related properties
- The main purpose is to predict reliability and availability from the architectural models, before actual system implementation
- Covers the gap from requirements definition to quality analysis
- Provides methods and techniques for R&A prediction

Reliability **= probability of failure-free operation of a software system for a specified period of time in a specified environment**

Availability **= probability of a software system or service being available when needed**

RELEVANCE

- Faults and R&A problems can be detected before system implementation
  - modifications and corrections are easier and cheaper
- Applicability of architectural style can be detected before implementation
  - the architectural decisions can still be affected
- Different architectural solutions can be compared and the best possible candidate can be selected

EVALUATING EXECUTION QUALITIES – RAP method

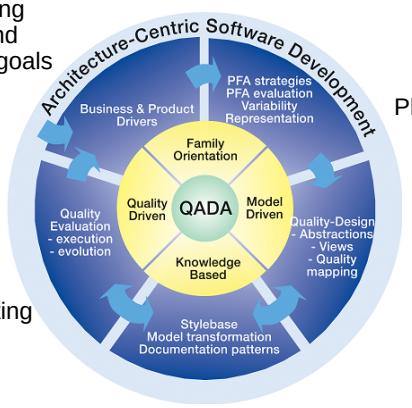
- RAP Introduction
- **Method overview**
- Phases
  1. Defining reliability and availability goals
  2. Representing reliability and availability in architectural models
  3. Evaluating reliability and availability
- Case example

RAP – OVERVIEW

- **Phases**
  - Three phases: 1) Defining reliability and availability goals, 2) Representing reliability and availability in architectural models, and 3) Evaluating reliability and availability.
- **Steps**
  - For each phase, a set of steps is defined.
- **Activities**
  - Steps can further include specific activities.
- **Views**
  - Three views of QADA, structural, behavior and deployment, are used in phases 2 and 3.
- **Evaluation levels**
  - The R&A evaluation is done incrementally according to four evaluation levels.

RAP as a part of QADA

Phase1: Defining reliability and availability goals



Phase 2: Representing reliability and availability in architectural models

Phase 3: Evaluating reliability and availability

EVALUATING EXECUTION QUALITIES – RAP method

- RAP Introduction
- Method overview
- Phases
  1. Defining reliability and availability goals
  2. Representing reliability and availability in architectural models
  3. **Evaluating reliability and availability**
- Case example

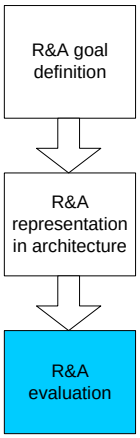
Phase 3: EVALUATING RELIABILITY AND AVAILABILITY

Purpose: to validate whether or not the R&A goals are met in the architecture.

Results: result of the R&A analysis.

Steps:

1. Quantitative analysis
2. Qualitative analysis
3. Decision making



Step 1: QUANTITATIVE ANALYSIS

Purpose: to calculate the reliability of the system as probability of failure of its components

Results: estimated probability of failure of the system and its components

Activities:

- Estimate component and connector reliability
- Estimate software system reliability
- Estimate system reliability

Step 1: ACTIVITIES

Estimate component and connector reliability as

Independent element

- Estimate the probability of failure of an independent component using
  - a Markov chain model, or
  - documentation
- Refine the achieved value with other properties of a component
  - E.g. component size/estimated size, (planned) implementation technology, (planned) fault tolerance, etc.
- Estimate the probability of failure of the connectors
  - Basing on the type of connection, interfaces, etc.

Case example

Step 1: ACTIVITIES

Estimate component and connector reliability as

Dependent element

- Simulate the system
  - choose the elements for simulation
  - define input messages
  - create a simulation model and run the simulation
- Basing on the results of simulation and the estimated probability of failure of independent elements, define and calculate
  - the probability of failure of components and connectors in each system execution path
  - the probability of failure of components and connectors in all execution paths (i.e. refined reliability of components and connectors in system execution)

Case example

Case example

Step 1: ACTIVITIES

Estimate software system reliability

- Compute the reliability of individual paths
  - Path reliability is the specified reliabilities of components and connectors involved in a path
- Calculate the software system reliability
  - The reliability of the software is a weighted average of reliabilities of all paths

Estimate system reliability

- Determine the reliability of the hardware
  - from previous use (experiences) or testing
- Define the reliability of hardware/software component combination
- Define the reliability of the network (between nodes)

Step 2: QUALITATIVE ANALYSIS

Purpose: to analyze whether or not the non-numerical requirements are met

Results: analysis report on how the architecture meets the requirements

Activities:

- Track the R&A requirements to architecture
- Track the architectural properties to the requirements
- Compare the design decisions with the R&A requirements and analyze how the requirements are met
- Identify problems that may occur when certain R&A requirements are not met

Step 3: DECISION MAKING

Purpose: to define whether or not the requirements are met well enough

Results: the decision to move to the next evaluation level or go back to the phase 2 to revise the architecture

Activities:

- Accept the architecture, or
- Revise the architecture by
  - Decrease the probability of failure of components and their interactions
    - choosing components with higher reliability (if available)
    - implementing higher reliable components by eliminating software defects in their implementation
    - deploying software on more reliable hardware.
  - Change the architecture by
    - changing styles and patterns
    - introducing new mechanisms (e.g. fault tolerance or fault treatment)

Case example

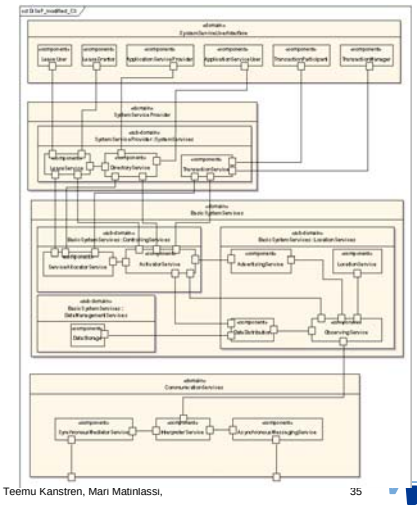
EVALUATING EXECUTION QUALITIES – RAP method

- RAP Introduction
- Method overview
- Phases
  1. Defining reliability and availability goals
  2. Representing reliability and availability in architectural models
  3. Evaluating reliability and availability
- **Case example**

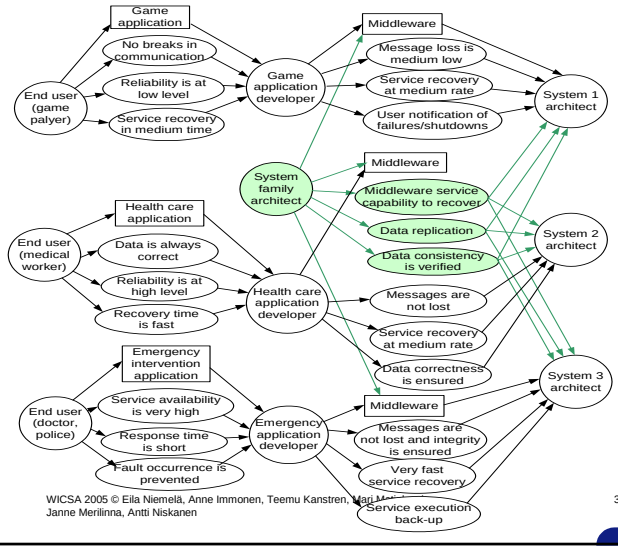
Continue

Case example: A Distributed Services Platform (DiSeP)

- DiSeP system family provides a distribution platform for a family of software systems.
- Includes three family members: middleware systems for game, health care and emergency intervention applications.



DiSeP - Phase 1: Identifying stakeholders and their concerns



DiSeP - Phase 1: Refining quality requirements

• Refined R&A requirements from the family architect's point of view

Req ID	Requirement description	Stakeholder	Importance
R2.1	**Middleware services are able to recover**	**System family architect**	**high**
R5	**Data consistency is verified in every 5 seconds**	**System family architect**	**low**
R6	**Data is replicated at least in 2 data storages**	**System family architect**	**medium**
R7	**Data may not be lost in failure/error situations**	**System family architect**	**medium**

Step description

DiSeP - Phase 1: Refining quality requirements

System	Functionality	R&A importance
S1: A middleware for game application	Light functionality	Low
S2: A middleware for health care application	Restricted functionality	Medium
S3:A middleware for emergency intervention application	Full functionality	High

DiSeP - Phase 1: Mapping R&A requirements to functionality

- Mapping family-specific R&A requirements to functionality

R&A requirement	Corresponding service
R2.1	**All the involved basic, system and communication services**
R5	**Data distribution**
R6	**Data distribution, Location service**
R7	**Data distribution**

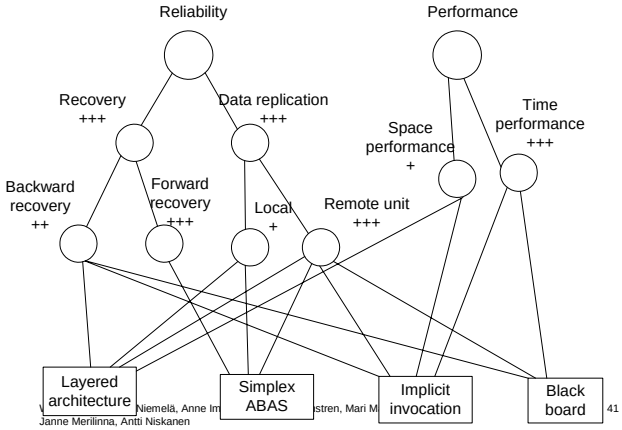
Step description

DiSeP - Phase 1: Mapping R&A requirements to functionality

Service	**Responsibility**	**Family-specific R&A requirement**	**System-specific R&A requirements for S3**
Data distribution	Contributes to the operation of distributed data storage. Creates, maintains and tracks connections to other units in order to share data. Allows data to be stored in local resources. Negotiates about the copying, transferring or deleting data if necessary.	R2.1, R5, R6	R7R1-S3, R2.2-S3, R4-S3, R8-S3
Location service	Sends after the given time period a notification signal about the existence of the node in the network. Maintains the location map of the network. Sends a signal to the user services of the own node to start registration when first time connected to the network. Announce the availability of the system services	R2.1, R6	R1-S3, A1-S3, R2.2-S3, A3-S3, R8-S3
Advertiser	Informs the active system service provider the availability of the user services of the own node.	R2.1	R1-S3, R2.2-S3
Observer	Routes messages from network to listeners and forward asynchronous messages. Routes outgoing messages to the network.	R2.1	R1-S3, R2.2-S3, R4-S3

DiSeP - Phase 1: Selecting an architectural style and doing the trade-off analysis

- **NFR framework for detecting conflicts**



Step description

DiSeP - Phase 1: Defining criteria for R&A evaluation

Criteria for evaluation of the DiSep system family

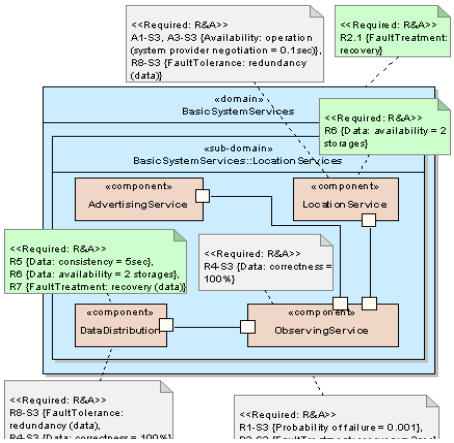
Evaluation level	Evaluation criteria	Corresponding requirement
Level 1	**System family-specific requirements**	**R2.1, R5, R6, R7**
Level 2	**High level system-specific requirements**	**A1-S3, A2-S3, R1-S3, R3-S3, R4-S3, R8-S3, R9-S3**
Level 3	**Medium level system-specific requirements**	**R2.2-S3, A3-S3, A4-S3**
Level 4	**Low level system-specific requirements**	**-**

Evaluation criteria	Req.ID	Importance	Impacted architectural elements
Service capability to recover	**R2.1**	**medium**	**All basic, system and communication services**
Data consistency verification	**R5**	**medium**	**Data distribution**
Data loss prevented in error situations	**R7**	**medium**	**Data distribution**
Data replication	**R6**	**low**	**Data storage, data distribution, location service**

Step description

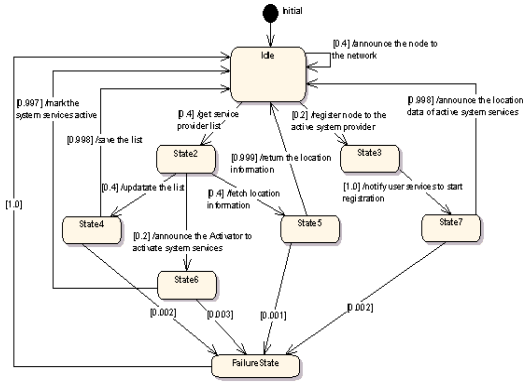
DiSeP - Phase 2: Representing required R&A in conceptual architecture

Service	Family-specific requirement	System-specific requirements
Data distribution	**R2.1, R5, R6**	**R7R1-S3, R2.2-S3, R4-S3, R8-S3**
Location service	**R2.1, R6**	**R1-S3, A1-S3, R2.2-S3, A3-S3, R8-S3**
Advertiser	**R2.1**	**R1-S3, R2.2-S3**
Observer	**R2.1**	**R1-S3, R2.2-S3, R4-S3**



DiSeP - Phase 3: Quantitative analysis

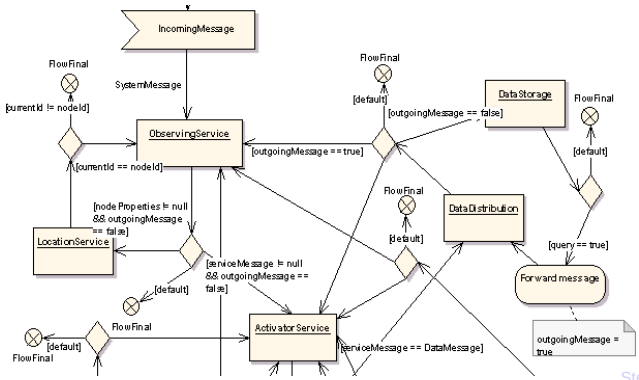
- Estimating reliability of an independent component using Markov chain model.



Step description

DiSeP - Phase 3: Quantitative analysis

- Simulating the system at the architecture level



Step description

DiSeP - Phase 3: Quantitative analysis

- **Predicted probability of failure of components of the system (in system execution), based on**
  - **estimated probability of failure of components, and**
  - **simulation**

Comp.ID	Component	Accessed	Probability of failure
C1	**Application Service Provider**	**1**	**0,000275**
C2	**Activator service**	**5**	**0.005**
C3	**Data storage**	**3**	**0.00075**
C4	**Directory service**	**1**	**0,000125**
C5	**Data distribution**	**5**	**0.001**
C6	**Observing service**	**8**	**0,00075**

Step description

DiSeP - Phase 3: Quantitative analysis

- **calculating the probability of failure for an execution path:**
  - ***Probability of failure (P1) = 1 - ((1 - C6)*(1 - ConC6C2)*(1 - C2) *(1 - ConC2C5)*(1 - C5) *(1 - ConC5C3)*(1 - C3) *(1 - ConC3C5)*(1 - C5) *(1 - ConC5C6)*(1 - C6)) = 0,.0096***
- **calculating the probability of failure of software system as a weighted average of execution paths:**
  - ***Probability of failure (system) = Probability of failure(P1)* Path probability(P1) + Probability of failure(P2)* Path probability(P2) + Probability of failure(P3)* Path probability(P3) =*** *0,0073*

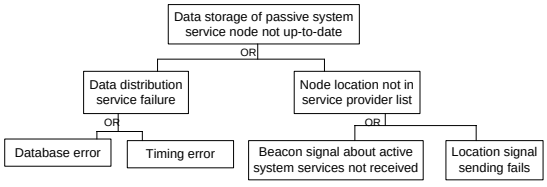
Step description

DiSeP - Phase 3: Qualitative analysis

- **requirements tracking**

R&A requirement	Conceptual level	Concrete level
R5: Data consistency is verified in every 5 seconds	**Data distribution service negotiates about data copies, transfers and deletions with other units.**	**Data distribution component includes a timer that starts data copying procedure every 5 seconds in the node of active system services.**
R6: Data is replicated at least in 2 data storages	**Each node includes a data storage that is continuously updated by the data distribution component. Location service of each node maintains the list of system services independently.**	**Each node includes a data storage that is continuously updated by the data distribution component. Location service of each node maintains the list of system services independently.**

- **identification of possible problem of the unmet requirement**



Step description

DiSeP - Phase 3: Decision making

- Reliabilities of components are satisfactory
  - Observing service and Activator service are the most critical components of the system
  - Activator service has the highest probability of failure value
- Numerical value for the probability of failure of the software system is 0.0073. The required probability of failure was max 0.01, thus the requirement R1-S3 is met in the architecture.
- Qualitative analysis proved that the requirements have been taken account in the architecture in a satisfactory manner.

=> the architecture is accepted!

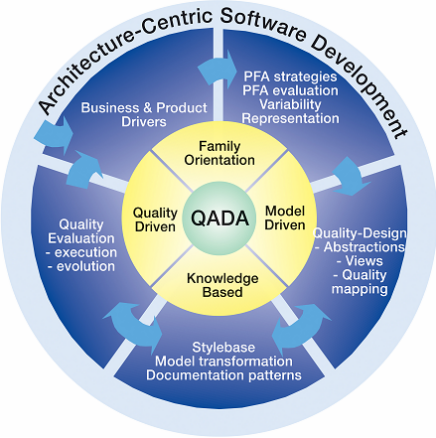
Step description

TUTORIAL OUTLINE

- INTRODUCTION
  - Main concepts of QADA®
- CAPTURING AND MAPPING QUALITY REQUIREMENTS TO ARCHITECTURE
- REPRESENTING QUALITIES IN ARCHITECTURE DESIGN
- EVALUATING EXECUTION QUALITIES
  - RAP method
- EVALUATING EVOLUTION QUALITIES
  - IEE method
- (RE)USING EXISTING DESIGN KNOWLEDGE
- CONCLUSIONS

EVALUATING EVOLUTION QUALITIES – IEE (Integrability & Extensibility Evaluation) method

- IEE Introduction
- Method overview
- Phases
  1. Defining IE Requirements
  2. Scenarios description
  3. IE Evaluation
- Case example



IEE INTRODUCTION

- IEE method = A scenario based method for Integrability and Extensibility Evaluation (IEE) at the architectural level.
- Covers the design activities from specifying, modelling and evaluating of quality properties.
- Intended for being used by architects
  - cost-effectively (i.e. the use takes only some hours) and
  - repeatedly (i.e. the method is easy to use).
- Suitable fro product family architectures and single system architectures.

Integrability is the ability to make **separately developed** components of a system to work correctly together.

Extensibility is the ability to **extend** a software system **with new features/services/components without loss of functionality or qualities** specified as requirements.

RELEVANCE

- Software family architecture that supports integrability and extensibility assists in:
  - Using 3rd party components.
  - Estimating adaptation required for a software family architecture or a component when components or services are renewed or new ones are added to the family.
  - Developing long-lasting software family architecture that will give better return on investment.
- The IEE method helps in achieving these goals.

EVALUATING EVOLUTION QUALITIES – IEE method

- IEE Introduction
- **Method overview**
- Phases
  1. Defining IE Requirements
  2. Scenarios Description
  3. IE Evaluation
- Case example

IEE METHOD OVERVIEW

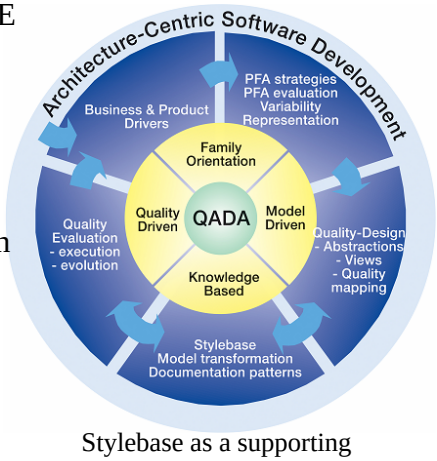
- **Phases**
  - Three phases: Quality requirements specification, scenario descriptions and evaluation
- **Steps/Activities**
  - For each phase, a set of steps with several activities and guidelines are defined.
- **Views**
  - QADA views - structural, behaviour, development and deployment - are in use.
- **Scenarios**
  - Modelling and evaluation is scenario based, i.e. iterative and incremental.
- **Iterations**
  - Modelling and evaluation are iterated based on priorities set by quality goals definition.
- **Knowledge base**
  - Stylebase is used as a supporting tool to find and evaluate the use of patterns for IE purposes.

IEE as a part of QADA®

1. Defining IE requirements

2. Scenarios description

3. IE Evaluation



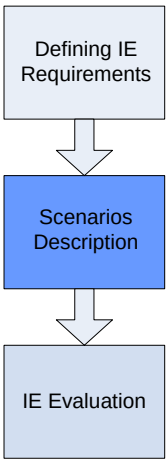
Stylebase as a supporting design and evaluation tool

ARCHITECTURAL VIEWS

- QADA uses four views:
  - structural, behavioral, development and deployment
- Views are presented on two abstraction levels:
  - conceptual and concrete.
- IEE method uses the views in modelling architecture according to the identified scenarios.
- Each scenario is represented in the views and abstractions that are affected by that scenario.
- Only the affected parts of the view are modelled in scenarios.

EVALUATING EVOLUTION QUALITIES – IEE method

- IEE Introduction
- Method overview
- Phases
  1. Defining IE Requirements
  2. **Scenarios Description**
  3. IE Evaluation
- Case example



Step 1: IDENTIFY SCENARIOS

Purpose is to identify the scenarios that are relevant to integrating and extending the architecture.

Result is a list of scenarios to be used as a basis for the evaluation.

Guidelines

- Identify scenerios belonging to the categories of IE scenarios:
  - Replacing existing services/components/technology platforms.
  - Adding new services/components/subsystems.
  - Adding new features to existing services/components.
- Use information from phase 1 to help identify the scenarios:
  - IE quality requirements and goals.
  - Variability.
- Consider the evaluators needs for evaluating the requirements.

case example

Step 2: DESCRIBE AND MODEL SCENARIOS

Purpose is to describe and model the required information in the identified IE scenarios in such a way that evaluation of the IE requirements can be performed.

Results are the descriptions and models of the scenarios that will be used to evaluate the defined IE requirements.

Guidelines

- Use UML 2.0 models and textual descriptions
- Describe what the scenario is, what components are involved and how are they affected.
- Use the stylebase to look for patterns and solutions.
- Define/refine assumptions, architectural constraints and design rationale of each view to document all used patterns and solutions.
- Consider the evaluators needs for evaluating the scenarios.

case example

Step 3: DEFINE THE REQUIRED COMPONENT TRACES

Purpose is to define which components need tracing and what kind of information should be traced for each component.

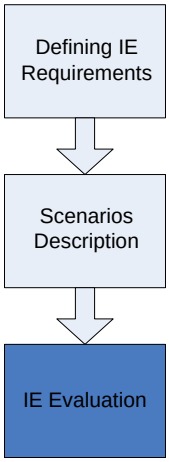
Result is a list of the required traces for components.

Guidelines

- Tracing of components is needed if not all information is available.
- Or when more detailed knowledge about component states and operations are required to ensure their compatibility.
- Requires that the component has built in support for tracing.
- Different types of traces:
  - Operational traces - Interaction of component operations
  - State traces – Object and data states in components/services

EVALUATING EVOLUTION QUALITIES – IEE method

- IEE Introduction
- Method overview
- Phases
  1. Defining IE Requirements
  2. Scenarios Description
  3. **IE Evaluation**
- Case example



Phase 3: IE EVALUATION

Purpose is to evaluate how the IE requirements are met in the architecture.

Result is a report of the evaluation results.

Steps for IE evaluation

1. Map scenarios to quality requirements
2. Evaluate the requirements
3. Compare evaluations results with the targets of the quality evaluation
4. Identify conflicts and report improvements
5. Report evaluation results.

Activities for evaluating the scenarios

- Architectural mismatch analysis
- Dependency analysis
- Extensibility analysis
- Simulation with instrumented components

Step 1: Map scenarios to requirements

Purpose is to map the scenarios to requirements so the requirements can be evaluated through the scenarios.

Guidelines

- Map the scenarios that contribute something to that requirement in the architecture.
- One scenario can be mapped to many requirements.
- Don't Repeat Yourself - Some of the results information can be reported elsewhere in another form.

Result is a mapping of scenarios to requirements with the following information

- Which scenarios are related to which requirement
- What solutions are used in each scenario to achieve the requirement
- How well does the scenario meet the requirement
- (Reasoning why a scenario is relevant to a requirement)

case example

Step 2: Evaluate the requirements

Purpose is to evaluate how well the requirements are met in the architecture.

Result is the evaluation results that show how well the requirements are supported in the architecture (scenarios).

Guidelines

- Evaluation is based on the scenarios.
- Scenarios relevant to each requirement were mapped in the previous step.
- Apply the relevant activities to each scenario.
- Consider the scenarios related to the requirement being evaluated.
  - ➢A scenario can be considered from a different viewpoint for a different requirement.
- The defined activities are only guidelines, apply common sense and experience.
  - ➢The most important thing is to evaluate how the requirement for integrability or extensibility is supported.

Step 3: Compare evaluation results with the targets of the quality evaluation

Purpose is to check how well the IE requirements are met in the architecture.

Result is a report of

- How well are each of the requirements met
- What solutions are used in the scenarios for achieving the requirement
- What requirements levels are met
- How serious are the conflicts

Guidelines

The evaluation report should answer at least the following questions:

- Are the IE quality requirements met in the scenarios?
- How serious are the possible conflicts or problems?
- Which level of requirements are met?
- What is the overall result of the evaluation?

Step 4: Identify conflicts and report improvements

Purpose is to identify and evaluate possible conflicts in achieving the IE qualities and to propose improvements to found conflicts.

Result is a report of the
- Identified conflicts
- Improvement suggestions for fixing the conflicts
- Unsolved problems

Guidelines
- Check the scenarios and requirements for conflicts.
  - Do any scenarios, solutions or requirements conflict with each other?
- Evaluate the impact of conflicts to each others (trade-offs)
  - Does fixing one affect another scenario/requirement?
- Propose improvements by using patterns and identify unsolved problems.
  - Use the stylebase as a guide for patterns and solutions.

case example

Step 5: REPORT EVALUATION RESULTS

Purpose is to report the results of the evaluation done in steps 1-4.

Result is a report telling
- How well are the requirements met
- Proposed improvements
- Unsolved problems

Guidelines
- Collect data from the previous step into a summary report of the results

Case example

Evaluation activities

Purpose is to provide a set of evaluation activities to use to help evaluate the scenarios**.**

Result from each activity is the analysis of a given aspect of quality in a scenario.

Activities defined for evaluating the scenarios:

1. Architectural mismatch analysis
2. Dependency analysis
3. Extensibility analysis
4. Simulation with instrumented components

Guidelines

Keep in mind the requirement and evaluate the scenarios not only based on the given activities but by what is relevant for the scenario and requirement.

Activity 1: Architectural mismatch analysis

Purpose is to check that the integrated components are compatible with their interfaces and assumptions about the architectural style of the system.

Result is a comparison of

- component features and the architectural styles of the system.
- component interfaces

Guidelines

- Applied when new components are added or existing ones are updated.
- Identify the main architectural style(s)
- Check that new components are compatible with the used architectural style(s).
  - Compare component features to styles and patterns
- Check interfaces and behaviour matches based on matching conditions

case example

Activity 2: Dependency analysis

Purpose is to check that when components or features have been added or replaced, appropriate techniques have been used to minimize dependencies.

Result is the analysis and description of how dependencies are minimized in the scenarios.

Guidelines to check

- Interfaces
  - When components need to be replaced
  - Matching conditions are used to control component interfaces.
- Component coupling
  - When components are replaced or added.
  - Coupling is minimized, change is localized and ripple effect prevented.
- Encapsulation
  - When components/features are changed or added.
  - Separate techniques are used to achieve increased cohesion and deferred binding times for feature types, component types and abstract layers

case example

Activity 3: Extensibility analysis

Purpose is to check how the architecture supports extensibility where it is required.

Result is a list of defined extension points and analysis of how the architecture supports extensibility in these points.

Steps for evaluating extensibility

1. Identify extension points – the places in the architecture where it needs to support easy addition of functionality.
2. Check the use of extensibility patterns, e.g. observer, façade, selector, proxy, bridge, etc., in those extension points.
3. Check the use of other possible extensibility supporting mechanisms.
   - Use the stylebase as an on-line guide to assist in the evaluation of the use of patterns and solutions.

case example

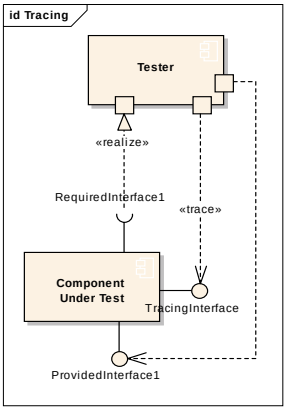
Activity 4: Simulation with instrumented components

Purpose is to gather the required information to evaluate the integrability of components.

Result is gathered data from the component traces.

Guidelines

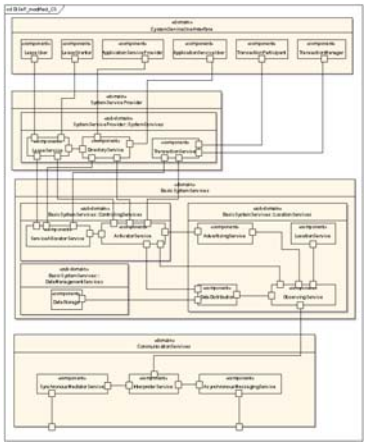
- The required component traces were defined in phase 2.
- To be able to trace the components, they must support the used tracing technique.
- Tracing is done using simulation through tester components that stimulate the component and gather data about its states and operations as related to the defined required component traces.



Go to next

Case example: A Distributed Services Platform (DiSeP)

- The DiSeP family provides a distribution platform for a family of software systems.
- The DiSeP family contains two products: Basic product A and Advanced product B.
- The architecture is service oriented, i.e. each product is built on top of the services provided by the DiSeP.
- The scope of family is limited to the platform services so that applications are considered only in application interfaces provided on top of platform services.



Tutorial outline

DiSeP - Phase 2: Identifying scenarios

From the requirements defined in phase 1 three different types of scenarios for integration and extension were defined:

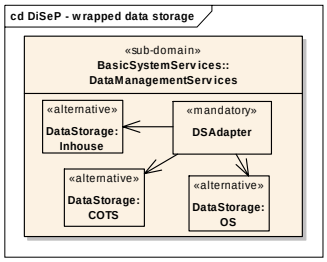
1. **Existing services in the product family are replaced with new ones.**
   - An inhouse component is replaced with a COTS, OS or OTS.
   - DiSeP is extended to a new software platform.
   - **…** Step description
2. **New features are added to existing services.**
   - A new feature is added to basic services.
   - A new application is integrated into DiSeP providing new features and services through the platform.
3. **New services are added to the product family**
   - A new communication protocol is added.
   - The location service (component) is added.
   - The transaction service (component) is added.

DiSeP - Phase 2: Describing scenarios

- **Description:** The data storage must support different variants of in-house, OS and COTS components. To support these requirements, we add a new component to the architecture that functions as an interface between the rest of the system and the data storage component. The responsibilities for this component include
  - 1) adapting the possible differences of the variants to work with the rest of the system and
  - 2) providing the rest of the system a unified interface to the used data storage component.
- **Other knowledge to document:** The used adapter component is considered to have properties from the adapter and facade patterns.

DiSeP - Phase 2: Modelling scenarios

- DataManagementServices domain is modelled to show the adapter style component and the alternative DataStorage components.
- Only this part is modelled for this scenario as it is the only affected part of the architecture.
- The variation modelling in DiSeP follows the notation of the PFE profile defined for product family engineering
- DSAdapter is always present and connects to the chosen alternative data storage component.
- DSAdapter is the interface component between the rest of the system and the data storage component.



Step description

DiSeP - Phase 3: Mapping scenarios to requirements

Requirement	Scenario	Reasoning for mapping
I2: Diversity of languages and component models	**2: A new application is integrated into DiSeP.**	**The scenario describes how new applications can be implemented on different platforms and different programming languages. This makes it directly related to supporting diverse programming languages.**
	4: Different component models.	**The scenario describes the integrability of components using different component models and is thus directly related to this requirement.**
	5: Different implementation languages	**The scenario describes the integrability of components implemented using different programming languages and is thus directly related to this requirement.**
I3:Substitutability of middleware services	**1: Replacing existing services.**	**The scenario considers replacing the data storage service which is a middleware service and has alternative variants. Thus it relates to substitutability of middleware services.**
	7: Adding the transaction service.	**The scenario concerns the transaction service which is a middleware service. The scenario also describes the services alternative variants and thus the services substitutability.**

Step description

DiSeP - Phase 3: Evaluating the requirements

- We start with the first requirement - I1: Style conformance
- The requirement states that new components must comform to the architectural style of the system.
- Evaluate the scenarios that were mapped to the I1 requirement with the relevant evaluation activities.
- The requirement concerns scenarios that add or replace components
  - ➢This includes most of the scenarios in DiSeP
- For each scenario that is relevant we check that they maintain the architectural style
  - ➢Architecture mismatch analysis

Step description

DiSeP - Phase 3: Compare evaluation results with the quality goals

- Two important requirements are partly met
- Two less important quality requirements were not supported
- All other requirements are met

- Most important level of requirements is partly met
- Second most important level is met
- Third and fourth level of requirements have some unsupported requirements and are thus not met

- The identified conflicts are not considered serious for the product family.

Step description

DiSeP - Phase 3: Identify conflicts and report improvements

- DiSeP has two important requirements that were evaluated to partly met
  - I1: Style comformance
  - I2: Heterogeneity of languages and component models is managed
- This is because a change of the implemention language of a component can cause changes to two components on two layers
- Improvement suggestion: Use a single component model that allows
  - components written in different programming languages to be integrated
  - wrapping components using other component models to the used system component model
- This allows the use of different programming languages and component models while localizing the change

Step description

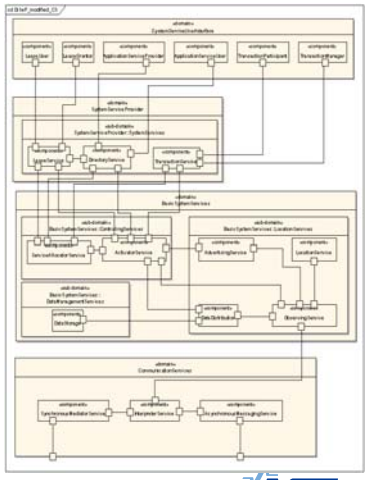
DiSeP - Phase 3: Report evaluation results

- The level of requirements met:
  - Most important level is partly met
  - Second most imporatl level is met
  - Third and fourth level are not met
- The most important level is partly met because a scenario integrating components in different programming languages can cause changes to two components in two layers.
- **Improvement suggestion:** Use a single component model to localize change and meet the requirements
- **Unsolved problems:** Runtime changes and doubled system services are not considered in the current architecture which makes the two least important levels of requirements not met
- The identified **conflicts** are not considered serious for the product family

Step description

DiSeP - Phase 3: Architectural mismatch analysis; styles

- **Example scenario: Existing in-house services are replaced with OS or COTS.**
- The main architectural style is identified as the ”Layers” architectural style where the domains are the layers.
- A COTS database typically functions in a client-server style. The in-house data storage system is a simple data structure object for storing data. Thus they have different architectural styles and interfaces.
- In the example scenario the style and interface differences are adapted by using an adapter component. The change of component is not visible to any components connected to it.
  - ➢Mismatch is avoided.



DiSeP – Phase 3: Architecture mismatch analysis; interfaces

The interfaces must be checked to see that the new component can provide all the services that the old component provides.

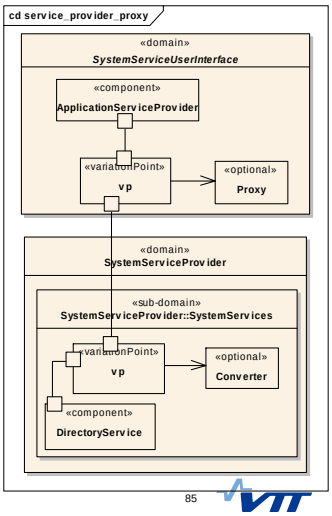
The scenario considers replacing the inhouse component interface with the MySQL JDBC interface and thus its services are mapped to the services of the in-house data storage. The mapping shows that all services can be provided.

In-house	MySQL	Rationale
storeData	**executeUpdate**	**Both store data and the parameters for storeData can be encoded in the SQL query for the database.**
getData	**executeQuery**	**Both retrieve data and the parameters for getData can be encoded in the SQL query for the database.**
searchService	**executeQuery**	**This is a special case of the getData service and can be handled with the same service and some extra processing.**

Activity description

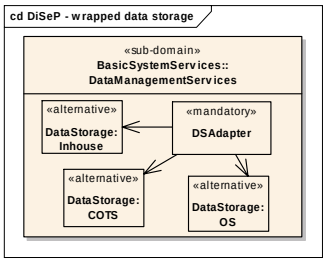
DiSeP - Phase 3: Dependency analysis

- **Scenario:** Components written in different programming languages need to interact.
- Scenario describes using a proxy-converter pair when needed to make components work together.
- Changing one of the components to a different programming language causes changes to two components.
- This causes a ripple effect and changes to two layers, making them more tightly coupled. This is clearly not an optimal solution.
- What can be done to improve?
- Use the stylebase to look for another pattern that could fix the problem -> suggestion: Use a single component model.



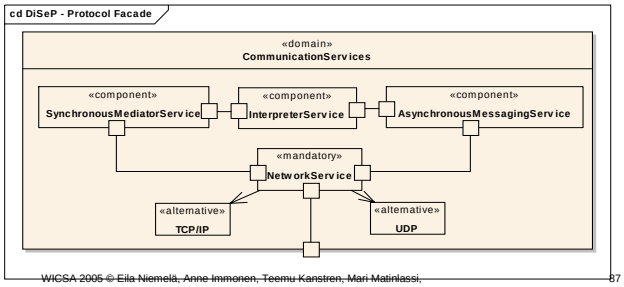
DiSeP - Phase 3: Extensibility analysis

- The most important DiSeP extensibility requirement – E4: Component extensibility
  - Extensibility of the architecture where it is required should be as easy as possible.
- Identified extension points in DiSeP:
  - ➢Transaction service needs to support different variants.
  - ➢Data storage service needs to support different variants.
  - ➢New protocols need to be supported by the communication services.
  - ➢It must be possible to extend the DiSeP platform to new programming languages and platforms.



DiSeP - Phase 3: Analysis of an extension point

- Communication protocols are used through the NetworkService component.
- The scenario describes this as a use of the facade pattern.
- The interface to the NetworkService component is always the same regardless of which protocol is used to communicate with other nodes.
- Adding new protocols only requires changes to the NetworkService component. Thus extensibility by adding support for new protocols easily is supported and the appropriate patterns are used.



Activity description

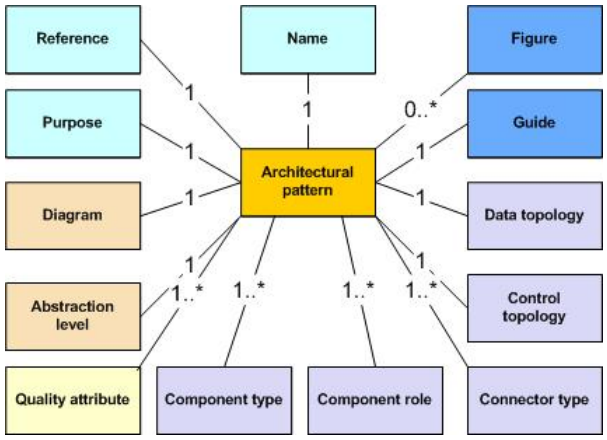
TUTORIAL OUTLINE

- **INTRODUCTION**
  - **Main concepts of QADA®**
- **CAPTURING AND MAPPING QUALITY REQUIREMENTS TO ARCHITECTURE**
- **REPRESENTING QUALITIES IN ARCHITECTURE DESIGN**
- **EVALUATING EXECUTION QUALITIES**
  - **RAP method**
- **EVQLUATING EVOLUTION QUALITIES**
  - **IEE method**
- **(RE)USING EXISTING DESIGN KNOWLEDGE**
- **CONCLUSIONS**

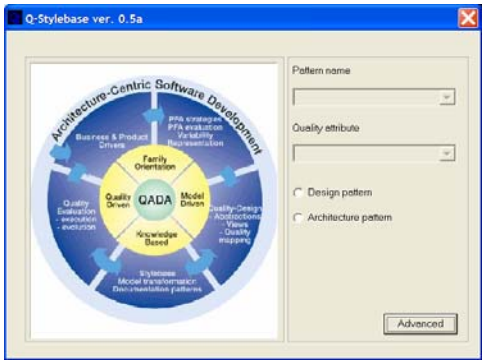
QADA TOOLING SUPPORT

- Background:
  - 'Stardardized' documentation
    - A pattern for component documentation
    - A pattern for architecture documentation
  - Model-Driven Development
  - Quality-Driven Architecture Development
- Q-Stylebase
- Q-Tra tool extension

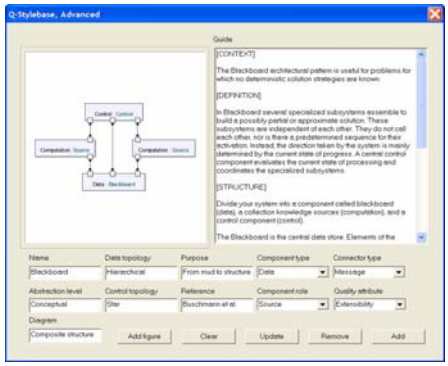
Q-Stylebase – Pattern Repository



Q-Stylebase - User Interface



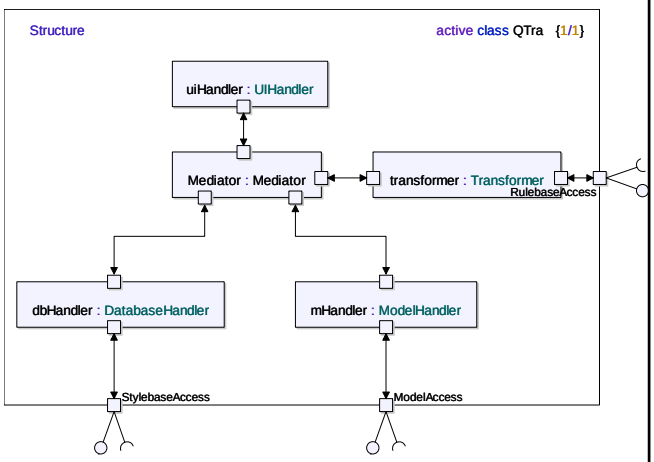
Query dialog



Management dialog

Q-Tra Tool

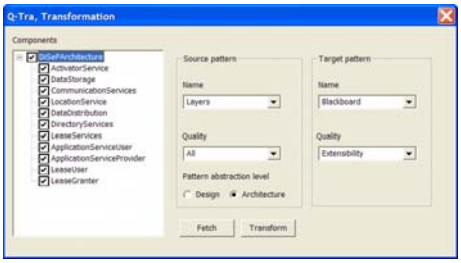
- Stylebase
  - Distributed database
    - Q-Stylebase
- Model handler
  - Access the modeling tool
- Transformer
  - Performs transformations
  - Rulebase
    - Q-RDL
- Graphical user interface
- Mediator
  - Provides loose coupling between components
    - Modifiable
    - Extensible



Q-Tra Tool

Q-Tra can be used in four ways

1. Electrical library for patterns
   - Using such as pattern catalogues
2. Quality-driven architecture model construction guide
   - Choosing the most suitable patterns for the problem by quality attributes
3. Quality-driven architecture model evaluation guide
   - Validating quality requirement by browsing the stylebase by pattern name
4. Quality-driven architecture model transformation tool
   - Transforms existing architecture to another one



TUTORIAL OUTLINE

- **INTRODUCTION**
  - **Main concepts of QADA®**
- **CAPTURING AND MAPPING QUALITY REQUIREMENTS TO ARCHITECTURE**
- **REPRESENTING QUALITIES IN ARCHITECTURE DESIGN**
- **EVALUATING EXECUTION QUALITIES**
  - **RAP method**
- **EVQLUATING EVOLUTION QUALITIES**
  - **IEE method**
- **(RE)USING EXISTING DESIGN KNOWLEDGE**
- **CONCLUSIONS**

CAPTURING AND REPRESENTING QUALITIES

QADA - QRF

- aims to enable quality evaluation at an early phase of software development
- includes steps and techniques for eliciting quality requirements, and transforming and modeling them in product family architecture
- is suitable for product families, allowing to manage variable requirements
- helps in evaluating that architecture meets the requirements
- has been applied to two experiments (laboratory and industrial)

EVALUATING EXECUTION QUALITIES

- QADA - RAP is a method for predicting reliability and availability at the architectural level
- Consists of three phases:
  1) Defining reliability and availability goals,
  2) Representing reliability and availability in architectural models by using RA profiles, and
  3) Evaluating reliability and availability from architecture
- Aims to improve reliability and availability of the product family enabling quality prediction in early development phase
- Tool support for evaluation has been developed as an add-on to a commercial tool

EVALUATING EVOLUTION QUALITIES

- QADA – IEE is for evaluating integrability and extensibility at the architectural level
- Consists of three phases:
  1. Defining IE Requirements
  2. Scenarios Description
  3. IE Evaluation
- Modelling and evaluation are scenario based with a set of steps and activities defined for each phase.
- Aim is to evaluate quality at an early phase with minimal time investment
- Evaluation is supported by a stylebase including a set of styles and patterns

REFERENCES – QADA®

- Niemelä, E. Strategies of Product Family Architecture Development. To appear in SPLC 2005, 12 p
- Niemelä, E., Matinlassi, M., Taulavuori, A. Practical Evaluation of Software Product Family Architectures, The 3rd International Conference on Software Product Lines, SPLC3, August- September 2004, 130-145.
- Matinlassi, M.. Comparison of software product line architecture design methods: COPA, FAST, FORM, KobrA and QADA. - Proceedings of the 26th International Conference on Software Engineering (ICSE 2004), Edinburgh, UK, 23 - 28 May 2004. (2004), 127 - 136.
- Dobrica, L., Niemelä, E. 2004. UML Notation Extensions for Product Line Architectures Modeling, The 5th Australasian Workshop on Software System Architectures (AWSA 2004), Melbourne, Australia, April 13-14, 2004, 44-51.
- Lago, P, Niemelä, E., van Vliet, H. Tool Support for Traceable Product Evolution, European Conference on Software Maintenance and Reengineering, CSMR, Tampere, Finland, March 24-26, 2004, 261-269
- Niemelä, E., Ihme, T. 2001. *Product Line Software Engineering of Embedded Systems.* Proceedings of SSR'01, Symposium on Software Reusability, Toronto, Ontario, Canada, May 18-20, 2001, pp. 118-125.
- **Niemelä, E., Kalaoja, J., Lago, P. Toward an architectural knowledge base for wireless service engineering. IEEE Trans. on Software Engineering,Vol. 31, No 5, May 2005, pp. 361-379**
- **Purhonen, A., Niemelä, E., Matinlassi, M. Viewpoints of DSP Software and Service Architectures. In the Journal of Systems & Software. 2004. Vol. 69, No. 1-2, 57-73.**
- **Merilinna, J., Matinlassi, M. Evaluation of UML tools for model-driven architecture. 11th Nordic Workshop on Programming and Software Development Tool and Techniques NWPER'2004. Turku, 17 - 19 Aug. 2004. TUCS General Publications (2004)**
- **Niemelä, E., Matinlassi, M., Lago, P. Architecture-centric approach to wireless service engineering. The Annual Review of Communications. International Engineering Consortium. Vol. 56. October 2003, 875-889. ISBN: 0-931695-22-9.**
- **Matinlassi, M., Niemelä, E, Dobrica, L. Quality-driven architecture design and quality analysis method. A revolutionary initiation approach to a product line architecture. Espoo, VTT Electronics, VTT Publications 456, 2002, 128 p. + 10 p. ISBN 951-38-5967-3; 951-38-5968-1.**
- **Merilinna, J. 2005. A Tool for Quality-Driven Architecture Model Transformation. Espoo, VTT Electronics. 106 p. + app. 7 p. VTT Publications; 561**
  ISBN 951-38-6439-1; 951-38-6440-5 http://www.vtt.fi/inf/pdf/publications/2005/P561.pdf

REFERENCES – QADA®

- Immonen, A., Niemelä, E., Matinlassi, M. Evaluating the integrability of COTS components - software product family viewpoint. In: Testing Commercial-off-the-Shelf Components and Systems, Beydeda, Sami; Gruhn, Volker (Eds.), Jan. 2005, Springer-Verlag, ISBN: 3-540-21871-8,. 141-168.
- **Merilinna, J. and Niemelä, E. A Stylebase as a Tool of Quality-Driven Software Architecture Modelling. In: Proceedings of the Ninth Symposium on Programming Languages and Software Tools, August 13-14, 2005, Tartu, Estonia. Pp. 97-111. ISBN 9949-11-113-7.**
- Immonen, A., A method for predicting reliability and availability at the architectural level. To appear in Research Issues in Software Product-Lines - Engineering and Management, Timo Käkölä and Juan Carlos Dueñas (Eds.), 2005, Springer.
- **Immonen, A. and Niskanen, A. A tool for reliability and availability prediction. Accepted to the 31th Euromicro Conference on Software Engineering and Advanced Applications. 2005, Porto, Portugal. 8 p.**
- Matinlassi, M. Evaluating the portability and maintainability of software product family architecture: terminal software case study. - Proceedings of the 4th IEEE/IFIP Conference on Software Architecture (WICSA), 12 - 15 June 2004 Oslo, Norway, Magee, J., Szyperski, C., Bosch, J. (Eds). IEEE Computer Society (2004), 295 – 298
- Matinlassi, M., Niemelä, E. The impact of maintainability on component-based software systems. The 29th Euromicro conference, Component-based software engineering track. Antalya, Turkey, 3-5 Sep. 2003, 25-32.
- Dobrica, L., Niemelä, E. A Survey on Software Architecture Analysis Methods. IEEE Transactions on Software Engineering, Vol. 28, No 7, July 2002, 638-653.
- Dobrica, L., Niemelä, E. A strategy for analysing product-line software architectures. VTT Espoo: Technical Research Centre of Finland, VTT Publications 427, 2000, 124 p. ISBN 951-38-5598-8
- Dobrica, L.; Niemelä, E. 2000. Attribute-based product-line architecture development for embedded systems. Proceedings of the 3rd Australasian Workshop on Software and Systems Architectures. Sydney, 19 - 20 Nov 2000. IEEE. US (2000), 76 – 88.
- See QADA also at http://www.vtt.fi/qada/

OTHER REFERENCES

- Pohl, K., Böckle, G., van der Linden, F. J. *Software Product Line Engineering. Foundations, Principles and Techniques*. 2005, XXVI, 468 p., Springer-Verlag, ISBN: 3-540-24372-0
- **van der Linden, F., Bosch, J., Kamsties, E., Känsälä, K., Obbink, H. Software Product Family Evaluation. SPCL 2004, Boston, USA, pp. 110-129**
- **Bosch, J. On the Development of Software Product-Family Components. SPLC 2004, Boston, USA, pp. 146-164.**
- **Bass, L., Clements, P., Kazman, R.. *Software Architecture in Practice*. Reading, Massachusetts: Addison-Wesley, 1998.**
- **Bosch, J. *Design and Use of Software Architectures - Adopting and Evolving a Product line Approach*, Addison Wesley, 2000. ISBN 0-201-67494-7.**
- **Hofmeister, C., Nord, R., Soni, D. *Applied software architecture*. Addison-Wesley, 1999.**
- **P. B. Krutchen. The 4+1 View Model of Architecture. IEEE Software, November 1995, pp. 42–50.**
- **Buhne, S., Chastek, G., Käkölä, T., Knauber, P., Northrop, L., Thiel, S.: Exploring the Context of Product Line Adoption. Lecture Notes in Computer Science, Vol. 3013. Springer-Verlag, Berlin Heidelberg New York (2003)**
- **Schmidt, K., Verlage, M.: The Economic Impact of Product Line Adoption and Evolution. IEEE Software,** 19 **(4), (2002), 50-57.**
- **Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Boston, MA, USA: Addison-Wesley. (2002)**
- **van der Linden, F., Bosch, J., Kamsties, E., Känsälä, K., Krzanik, L., Obbink, H.: Software Product Family Evaluation. Lecture Notes in Computer Science, Vol. 3014. Springer-Verlag, Berlin Heidelberg New York (2003), 376-394**
- **Obbink, H., America, P., van Ommering, R., Muller, J., van der Sterren, W., Wijnstra, J. G.: COPA: A Component-Oriented Platform Architecting Method for Families of Software-Intensive Electronic Products. SPLC1, (2000)**
- **America, P., Rommes, E., Obbink, H.: Multi-View Variation Modeling for Scenario Analysis. Lecture Notes of Computer Science, Vol. 3013. Springer-Verlag, Berlin Heidelberg New York (2003)**
- **Wijnstra, J.G.: Evolving a Product Family in a Changing Context. Lecture Notes of Computer Science, Vol. 3013. Springer-Verlag, Berlin Heidelberg New York (2003).**